**🡪 Wrapper Class: -**

Wrapper function in python calls another function when it is needed. When you want to modify a behaviour of a function you can use a wrapper around it. It is associated with the concept of decorators in python. When data scientists translate thoughts into codes many times they need to make changes in their existing function to match their new idea like adding arguments or a computational function at that times wrapper class comes to help them and it can be used as an interface to adapt to the existing codes and it saves from the trouble of modifying codes again and again.

* Surround Framework code: -

|  |  |
| --- | --- |
|  |  |
| ClassWrapper(): |  |
|  |  | def \_\_init\_\_(self, surround, type\_of\_uploaded\_object=None): |
|  |  | self.surround = surround |
|  |  | self.actual\_type\_of\_uploaded\_object = None |
|  |  | if type\_of\_uploaded\_object: |
|  |  | self.type\_of\_uploaded\_object = type\_of\_uploaded\_object |
|  |  | else: |
|  |  | self.type\_of\_uploaded\_object = AllowedTypes.JSON |
|  |  | self.surround.init\_stages() |
|  |  |  |

* As we can see in the above example in class wrapper its defined init function and it will used to call this function in the code and to modify it.

🡪 Code to import wrapper class: -

|  |
| --- |
| Import\_json |
|  | from surround import Surround, Wrapper, AllowedTypes |
|  | from stages import ValidateData, {project\_name}Data |
|  |  |
|  | class PipelineWrapper(Wrapper): |
|  | def \_\_init\_\_(self): |
|  | surround = Surround([ValidateData()], \_\_name\_\_) |
|  | super().\_\_init\_\_(surround) |
|  |  |
|  | def run(self, input\_data): |
|  | text = json.loads(input\_data)["data"] |
|  | data = {project\_name}Data(text) |
|  | self.surround.process(data) |
|  | return {{"output": data.output\_data}} |